Average cost of orthogonal range queries in multiattribute trees

Danièle Gardy, Philippe Flajolet, Claude Puech

INRIA

UNITE DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tel (1) 39 63 55 11

Rapports de Recherche

N° 892

AVERAGE COST OF ORTHOGONAL RANGE QUERIES IN MULTIATTRIBUTE TREES

Programme 1

Danièle GARDY
Philippe FLAJOLET
Claude PUECH

Septembre 1988

RR_0892

Average Cost of Orthogonal Range Queries in Multiattribute Trees

DANIÈLE GARDY*, PHILIPPE FLAJOLET† AND CLAUDE PUECH‡

Abstract. We give exact and asymptotic formulæ for the average search and access cost of orthogonal range queries on multiattribute trees and doubly chained trees. Our results are also valid for paginated trees, and permit to evaluate the memory costs of the trees.

Coûts moyens de recherche dans les arbres multiattributs

DANIÈLE GARDY, PHILIPPE FLAJOLET ET CLAUDE PUECH

Résumé. On étudie le coût moyen d'une requête par intervalles sur des arbres multiattributs et sur leur implémentation par des arbres doublement chaînés. Nous obtenons des expressions exactes et asymptotiques pour le coût de recherche dans l'arbre, le coût d'accès aux enregistrements et la place mémoire, pour des arbres paginés et non paginés.

* LRI, CNRS UA 410, Université Paris-Sud, 91405 Orsay, France

† INRIA, Rocquencourt, 78150 Le Chesnay, France

‡ LIENS, Ecole Normale Supérieure, rue d'Ulm, 75005 Paris, France

Average cost of orthogonal range queries in multiattribute trees

Danièle GARDY Philippe FLAJOLET Claude PUECH

Abstract

We give exact and asymptotic formulæ for the average search and access cost of orthogonal range queries on multiattribute trees and doubly chained trees. Our results are also valid for paginated trees, and permit to evaluate the memory cost of the trees.

Keywords: *doubly chained tree, multiattribute tree, multidimensional searching, orthogonal range query, partial match retrieval.*

1 Introduction

Multiattribute trees are one of the many structures proposed for database implementation. Their performance has been studied by several authors. R.L. Kashyap *et al.* give in [6] a statistical algorithm for estimating the performance of a partially specified query on a sample database represented by a multiattribute tree, which V. Gopalakrishna *et al.* [5] generalize to more general queries. Both assume known specific parameters on the tree: average number of sons of a node at a given level, attributes ranked according to their probability in a query.... Another study of multiattribute trees, under the name "compressed, fully transposed ordered files", is presented by P. Svensson in [8] He gives a formula for the average search cost of a random orthogonal range query on what we call here a "standard" multiattribute tree.

In this paper, we study the average cost of a random orthogonal range query on a multiattribute tree on n records. We derive a general formula for this cost, which can then be tailored to the cases of a single query, partially specified queries, or any model of the query space for which we can compute some very general parameters, and which can also be used to get the average memory requirements, or to compare different attribute orders. The cost is also given for a particular implementation of multiattribute trees, doubly chained trees, and extended to paginated trees, which allows us to characterise the effect of pagination on a random query. This work is a generalization of previous results by P. Flajolet and C. Puech, who give in [3] [7] a formula for the average search cost of a given partially specified query.

The plan of the paper is the following: we introduce our assumptions and notations in Section 2, and show on an example the kind of results our formalization allows us to obtain in Section 3. Sections 4 to 6 deal with the cost of a random query in a multiattribute or doubly chained tree. Finally, we give asymptotic costs for large sizes of domains in Section 7, and derive some consequences in Section 8.

2 Definitions and notations

A multiattribute tree can be defined as a directory to a file of n records; each leaf gives access to either a record or its memory address, and the path to a leaf is uniquely determined by the value of the associated record. Alternatively, it can be seen as a heterogeneous lexicographic tree, with a different set of admissible labels for each level.

The records are built on k attributes $A_1 \ldots A_k$, and are elements of the cartesian product $D_1 \times \ldots \times D_k$, where D_i is the domain of attribute A_i, and is of finite size d_i. Let us define $d_{>j} = d_{j+1}d_{j+2}\ldots d_k$ for $j \geq 0$ ($d_{>k} = 1$). Each node at level i in the tree has d_{i+1} potential sons, corresponding to the d_{i+1} possible values of attribute A_{i+1}. For example, the root has at most d_1 sons; those actually present in the tree built on a given file are the nodes labelled by a value x, such that there exists at least one record in the file, having value x on the first attribute.

We assume a uniform probability law on each domain: each value of D_i has probability $1/d_i$ of being selected. We also assume that the probability law on $D = D_1 \times \ldots \times D_k$ is the product of the probability laws on each D_i, i.e. the values taken by a k-tuple on different attributes are independent. With these requirements, each k-tuple of D has a probability $1/d$ of being selected, where $d = d_{>0} = d_1 \ldots d_k$.

We shall study two cases for a multiattribute or doubly chained tree:

- In the *standard* case, the tree is not paginated, and all nodes are present.

- The tree may be *paginated* (bucketed): we store any subtree containing at most b records in a single node.

The parameter b, representing the number of records which can be stored in a single leaf or in a single page of memory, is allowed to range in $[1 \ldots d_k - 1]$. The case $b = 1$ is equivalent to the case of a "pruned" tree (we do not store *in extenso* a subtree with just one leaf, but compress it into a single node). We usually do not store in a page more than $d_k - 1$ records (otherwise, the value of a record on the last attribute A_k would never be considered, in building the multiattribute tree!). The formulæ for a standard tree can be derived from the results for a paginated tree by substituting 0 for b. As a consequence and although a page size of 0 does not have an intuitive signification, we shall often abbreviate "standard tree" by "$b = 0$" in the following sections.

It is important to note that our definition of pagination is consistent with the structure of the tree: We do not store the first b records in the first page, then the next b in the second page, and so on. Rather, we consider in turn each subtree of the tree, in order of increasing distance from the root, and store it in a single page if it contains at most b records. This means that a page may be less than full, but the records in a page will have the same values on the first few attributes:. Such a clustering effect may prove useful in retrieving records specified by a query.

We next precise the set of queries we shall consider: We are interested in *orthogonal range queries*, i.e. queries of the type

$$x_1 \in [a_1 \ldots b_1] \wedge \ldots \wedge x_k \in [a_k \ldots b_k].$$

The cost of a given query on a multiattribute tree can be decomposed into two parts:

A_1	code-part 1	a ... z
A_2	code-part 2	a ... z
A_3	salary	1 ... 100
A_4	age	20 ... 60
A_5	region	1 ... 10
A_6	qualification	1 ... 20

Figure 1: Domains of attributes

- the (internal) *search cost* of traversal of the tree (selection of the relevant leaves);
- the (external) *access cost* to the selected leaves.

These two costs are independent of each other, and can be computed separately. We shall study the search cost, which is measured by the number of internal nodes visited, and the access cost, measured by the number of leaves selected, as functions of the number of records stored in the tree. We define the *size* of the tree to be the number n of records stored in it (this is not to be confused with the usual notion of size of a tree as the total number of its internal and external nodes). We denote by $c^k_{u,n}$ the search or access cost of the query u averaged over all possible trees of size n (k is used here to emphasize the dependency on the number of attributes). When the query is chosen at random in the space of all legal queries, we also define $c^k_n = \sum_{u\ query} \text{Proba}(u)\ c^k_{u,n}$.

3 Example

In this section, we show on an example how the formulæ of sections 4 to 6 may be used to compare the performance of several orderings of attributes, for a given query, or to evaluate the gain of paginating a multiattribute tree. We consider a relation on 6 attributes, part of a database describing a set of employees. With each of the employees is associated a tuple containing his (her) qualification, salary and age, the code of the project (on two letters) and the region where (s)he works. Each of the attributes takes its values in a finite domain (see fig. 1). We assume that all possible records are equally likely.

The relation can be stored in a tabular, or array, form (uncompacted), or compacted in a multiattribute tree. To build this tree, we may consider several different orderings of attributes, each of which may have a different memory cost, and different access and search costs. We shall study here only a few of them (cf. fig. 2).

We give in fig. 3 the average memory required to store the relation in a standard multiattribute tree ($b = 0$), measured as the number of internal nodes of the tree, for the permutations of fig. 2, and in fig. 4 the same parameter, for several choices of page size. This shows that pagination substantially reduces the number of internal nodes of the tree; even a page size of 1 (corresponding to the case of a pruned tree, where we do not store *in extenso* the path to each leaf, but just enough to separate each record) gives a noticeable gain. Fig. 5 gives the average number of pages required to store the records, for various sizes of pages. With the exception of σ_3, the number of pages is roughly the same for $b = 5$ and $b = 10$. This means that a large

σ_1	code-1	code-2	salary	age	region	qual.
σ_2	code-1	code-2	region	qual.	age	salary
σ_3	region	qual.	code-1	code-2	salary	age
σ_4	qual.	salary	region	code-1	code-2	age
σ_5	code-1	code-2	qual.	age	salary	region

Figure 2: Permutations used

	σ_1	σ_2	σ_3	σ_4	σ_5
n=100	419.45	418.74	388.86	418.19	419.16
n=200	799.97	797.17	733.69	810.35	798.77
n=500	1878.75	1861.61	1670.53	1957.09	1871.30
n=1000	3541.84	3475.11	3115.81	3782.60	3512.47

Figure 3: Memory size: number of internal nodes for different orderings of attributes and a standard tree of varying size ($b = 0$)

page size results in less than optimal occupation of memory, and does not significantly reduce the number of pages.

We have also compared the different permutations of fig. 2 for several queries, specified by the length of the interval they select on each attribute (see fig. 6). For example, query 2 specifies a project code, and leaves all other attributes unspecified; query 4 specifies half of the project code, the region and the qualification; queries 1,3 and 5 select various intervals on different attributes. We give in fig. 7 the average number of internal nodes visited, for a non-paginated tree on 1000 records and the queries defined in table 6. As we can expect, a given ordering of attributes performs best with the queries which are the most selective on the first attributes. Figures 8 and 9 show the effect on the search and access costs of paginating a multiattribute tree ordered according to permutation σ_1. We give for comparison purposes the cost for a non-paginated tree ($b = 0$); in this case the access cost is equal to the number of records which satisfy the query. The search cost is always decreased by paginating the tree, but the access cost varies in

	σ_1	σ_2	σ_3	σ_4	σ_5
b=0	3541.84	3475.11	3115.81	3782.60	3512.47
b=1	329.02	392.16	291.36	226.55	357.67
b=5	30.01	29.89	87.86	21.09	30.07
b=10	27.21	27.10	13.75	21.07	27.27

Figure 4: Memory cost: average number of internal nodes for a paginated tree built on 1000 records

	σ_1	σ_2	σ_3	σ_4	σ_5
b=1	1000	1000	1000	1000	1000
b=5	536.39	532.84	615.50	787.19	534.70
b=10	522.18	522.17	221.57	787.09	522.18

Figure 5: Memory cost: average number of pages for a paginated tree built on 1000 records

attribute	query 1	query 2	query 3	query 4	query 5
code-1	26	1	26	1	26
code-2	26	1	26	26	26
salary	20	100	10	100	10
age	10	40	40	40	40
region	10	10	10	1	1
qualification	20	20	1	1	5

Figure 6: Queries: length m_i of the interval selected on attribute A_i

permutation	query 1	query 2	query 3	query 4	query 5
σ_1	847.71	7.20	848.44	102.57	758.43
σ_2	2725.11	7.10	1578.61	26.04	692.05
σ_3	2315.80	247.61	121.24	3.55	57.12
σ_4	773.32	1823.43	20.81	46.61	33.11
σ_5	1812.91	7.16	652.33	27.78	1064.94

Figure 7: Search cost for different orderings of attributes and a standard tree of 1000 records

	query 1	query 2	query 3	query 4	query 5
b=0	847.71	7.20	848.44	102.57	758.43
b=1	322.77	2.45	322.06	13.61	322.05
b=5	29.81	2.01	29.82	2.11	29.81
b=10	27.01	2.00	27.02	2.00	27.01

Figure 8: Search cost for permutation σ_1 and a paginated tree built on 1000 records

	query 1	query 2	query 3	query 4	query 5
b=0	50	1.48	5	0.19	2.5
b=1	380.11	1.48	305.09	38.45	305.06
b=5	522.78	0.79	521.08	20.63	521.08
b=10	522.18	0.77	522.17	20.08	522.17

Figure 9: Access cost for permutation σ_1 and a paginated tree built on 1000 records

a less predictable way. A query may select on a paginated tree much more leaves than there are records actually satisfying the query, due to the loss of information induced by pagination: We do not store in the tree information on *all* the attributes. For example, fig. 9 shows that the average number of leaves retrieved, for permutation σ_1, is always several orders of magnitude greater than the actual number of records satisfying the query, except for query 2.

The problem of finding the best order, for a given probability law on the space of queries or the memory cost, can be solved using our formulæ. It naturally depends on the respective importance of the memory, search and access costs. As can be seen on our example, it may depend on the number of records, and the page size: The binomial coefficients do not vary uniformly for comparatively small n. For example fig. 3 shows that, for $n = 100$, permutation σ_4 builds a tree with less internal nodes that permutation σ_1, but this order is reversed for $n = 200$ or greater. Fig. 4 shows that, as regards the number of internal nodes required to store the tree,σ_3 is the best choice of the five permutations for a standard tree or a paginated tree with a page size of 10, but is outperformed by σ_4 for a pruned tree, or a page size of 5. Moreover, the great variance in access costs between queries means that, by optimizing the order of attributes for an abstract "average" query, we run a serious risk of thrashing for a query differing too much with this average value.

In general, one can see on this example, and this is corroborated by the asymptotic expressions of Section 7, that pagination always reduces the internal search cost of a random query, but may behave very poorly with respect to its external access cost. Again this phenomenon can be precisely quantified using results of the next sections. Our estimates can be used to determine the best possible implementation in a given context.

4 Average cost of searching in a multiattribute tree

We define the *profile* of a query u as the k-tuple $(m_1, \ldots, m_k)$, where each m_i is the size of the interval selected by u (for integer-valued domains $D_i = [1 \ldots d_i]$ on which u selects intervals $[a_i \ldots b_i]$, $m_i = b_i - a_i + 1$). For example, u specified on attribute A_i corresponds to $a_i = b_i$ and $m_i = 1$, while u unspecified on attribute A_i corresponds to $m_i = d_i$ (all values of D_i are admissible). We shall often denote by the same letter u either a query or a profile.

We specify now the kind of probability law on queries (profiles) we are interested in. We assume that the intervals on the D_i are selected independently by a query. For each i, $1 \leq i \leq k$, we define $p_i(m)$, the probability that an orthogonal range query selects the k-uples whose value on the i-th attribute lies in an interval of length m, i.e. the probability that the associated

profile has value m in place i: $\sum_{m=1}^{d_i} p_i(m) = 1$. With these assumptions, a query u of profile $(m_1, \ldots, m_k)$ has probability $\prod_{i=1}^{k} p_i(m_i)$. We denote by $\overline{m_i}$ the average length of the interval selected by a random query on the i-th attribute: $\overline{m_i} = \sum_{m=1}^{d_i} m\, p_i(m)$.

Theorem 1:
The average search cost of a random orthogonal range query on a multiattribute tree built on n *records is:*

$$c_n^k = 1 + \sum_{j=1}^{k-1} \overline{m_1}\ \overline{m_2} \ldots \overline{m_j}\ \alpha_j$$

where:

- *for a standard tree,* $\alpha_j = 1 - \frac{\binom{d-d_{>j}}{n}}{\binom{d}{n}}$;
- *for a paginated tree,* $\alpha_j = 1 - \sum_{p=0}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}}{p}}{\binom{d}{n}}$.

Proof of Theorem 1:
We first derive $c_{u,n}^k(t)$, the exact cost of running a query u on a fixed tree t on n records ($n \geq 1$), for a fixed profile u. By summing $c_{u,n}^k(t)$ on t, then on u, we get a recurrence relation between the c_n^k, which we use to get an exact expression of their generating function.

We decompose t into $(\bullet, t_1, \ldots, t_{d_1})$: the t_i are the subtrees of t (which may be empty) corresponding to the k-tuples whose value on the first attribute is the i-th of domain D_1. The size of t_i (number of k-tuples whose first value is i) is noted $|t_i|$. We can express $c_{u,n}^k(t)$ as a function of the costs on the subtrees t_i. The number of nodes traversed is 1 for the root, plus the number of nodes in each of the t_i where we continue the search:

$$c_{u,n}^k(t) = 1 + \sum_{relevant\ t_i} c_{u_{>1},|t_i|}^{k-1}(t_i)$$

where $u_{>1}$ is the query induced by u on the attributes $A_2 \ldots A_k$, and we sum over all the subtrees selected by the query at the first level. If u selects the k-tuples whose value on the first attribute belongs to the interval $[a_1 \ldots b_1]$, then the selected subtrees are $t_{a_1}, t_{a_1+1}, \ldots, t_{b_1}$:

$$c_{u,n}^k(t) = 1 + \sum_{i=a_1}^{b_1} c_{u_{>1},|t_i|}^{k-1}(t_i).$$

We obtain $c_{u,n}^k$ by averaging $c_{u,n}^k(t)$ over all trees t of size n:

$$\begin{aligned} c_{u,n}^k &= \sum_{t\ of\ size\ n} \mathrm{Proba}(t/n)\ c_{u,n}^k(t) \\ &= 1 + \sum_{t=(\bullet,t_1,\ldots,t_{d_1})} \mathrm{Proba}(t/n) \sum_{i=a_1}^{b_1} c_{u_{>1},|t_i|}^{k-1}(t_i) \end{aligned}$$

where the probability on the tree t is conditioned by its size n.

The cost of searching in a subtree depends on its size, not on its rank as son of the root, and we pursue the search in m_1 subtrees. By symmetry, we get:

$$c^k_{u,n} = 1 + m_1 \sum_{t=(\bullet,t_1,\ldots,t_{d_1})} \text{Proba}(t/n)\; c^{k-1}_{u_{>1},|t_1|}(t_1). \tag{1}$$

The probability of the tree $t = (\bullet, t_1, \ldots, t_{d_1})$, conditioned by the size n of t, and where the subtrees $t_1, \ldots, t_{d_1}$ have respectively $n_1, \ldots, n_{d_1}$ leaves, is:

$$\begin{aligned}\text{Proba}(t/n) &= \text{Proba}(|t_1| = n_1, \ldots, |t_{d_1}| = n_{d_1}/|t| = n)\; \text{Proba}(t_1/n_1) \ldots \text{Proba}(t_{d_1}/n_{d_1}) \\ &= \frac{\binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}} \times \text{Proba}(t_1/n_1) \times \ldots \times \text{Proba}(t_{d_1}/n_{d_1})\end{aligned}$$

We use this expression in equation (1) to get:

$$c^k_{u,n} = 1 + m_1 \sum \frac{\binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}} \sum \text{Proba}(t_1/n_1) \ldots \text{Proba}(t_{d_1}/n_{d_1})\; c^{k-1}_{u_{>1},n_1}(t_1)$$

where the first sum is taken for all tuples of integers $(n_1, \ldots, n_{d_1})$ satisfying $n_1 + \ldots + n_{d_1} = n$, and the second sum ranges on all trees $t = (\bullet, t_1, \ldots, t_{d_1})$ such that $|t_1| = n_1, \ldots, |t_{d_1}| = n_{d_1}$. This last sum is equal to:

$$\sum_{|t_1|=n_1} \text{Proba}(t_1/n_1)\; c^{k-1}_{u_{>1},n_1}(t_1) \sum_{|t_2|=n_2} \text{Proba}(t_2/n_2) \ldots \sum_{|t_{d_1}|=n_{d_1}} \text{Proba}(t_{d_1}/n_{d_1})$$

i.e. to

$$\sum_{|t_1|=n_1} \text{Proba}(t_1/n_1)\; c^{k-1}_{u_{>1},n_1}(t_1).$$

This gives for the average cost $c^k_{u,n}$:

$$c^k_{u,n} = 1 + m_1 \sum_{\substack{n_1 \ldots n_{d_1} \geq 0 \\ n_1 + \ldots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}} \sum_{|t_1|=n_1} \text{Proba}(t_1/n_1)\; c^{k-1}_{u_{>1},n_1}(t_1) \tag{2}$$

$$= 1 + m_1 \sum_{\substack{n_1 \ldots n_{d_1} \geq 0 \\ n_1 + \ldots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}}\; c^{k-1}_{u_{>1},n_1}. \tag{3}$$

The probability of a query (or profile) u selecting on the first attribute an interval of length m_1 is: $\text{Proba}(u) = \text{Proba}(m_1)\; \text{Proba}(u_{>1})$. We then sum equation (3) on u, and derive a similar relation on the c^k_n:

$$\begin{aligned} c^k_n &= \sum_{u\; profile} \text{Proba}(u)\; c^k_{u,n} \\ &= 1 + \sum_{u = m_1 \bullet v} m_1\; p_1(m_1)\; \text{Proba}(v) \sum_{\substack{n_1 \ldots n_{d_1} \geq 0 \\ n_1 + \ldots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}}\; c^{k-1}_{v,n_1} \\ &= 1 + \overline{m_1} \sum_v \text{Proba}(v) \sum_{\substack{n_1 \ldots n_{d_1} \geq 0 \\ n_1 + \ldots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}}\; c^{k-1}_{v,n_1} \end{aligned}$$

where v runs through the space of all profiles on $D_2 \times \ldots \times D_k$. We next eliminate the sum on v with the relation: $c_{n_1}^{k-1} = \sum_v \text{Proba}(v)\, c_{v,n_1}^{k-1}$ to get:

$$c_n^k = 1 + \overline{m_1} \sum_{\substack{n_1 \ldots n_{d_1} \geq 0 \\ n_1 + \ldots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}}\, c_{n_1}^{k-1}. \tag{4}$$

Let $C_k(z) = \sum_{n \geq 0} \binom{d}{n} c_n^k z^n$ and $f_k(z) = \sum_{n \geq 0} \binom{d}{n} 1_{\{c_n^k > 0\}}\, z^n$. The recurrence relation (4) between c_n^k and $c_{n_1}^{k-1}$ translates into the generating functions relation:

$$\begin{aligned} C_k(z) &= f_k(z) + \overline{m_1} \sum_n \sum_{n_1 \ldots n_{d_1}} \binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}} c_{n_1}^{k-1}\, z^n \\ &= f_k(z) + \overline{m_1} \sum_{n_1} \binom{d_{>1}}{n_1}\, c_{n_1}^{k-1}\, z^{n_1} \sum_{n_2} \binom{d_{>1}}{n_2}\, z^{n_2} \ldots \sum_{n_{d_1}} \binom{d_{>1}}{n_{d_1}}\, z^{n_{d_1}}. \end{aligned}$$

The first sum ranges from $n_1 = 1$ (standard case) or $n_1 = b + 1$ (paginated tree) to $d_{>1}$, and is equal to $C_{k-1}(z)$, the generating function of the average cost of a random query over $D_2 \times \ldots \times D_k$. The $d_1 - 1$ other sums range from $n_i = 0$ to $d_{>1}$, and are each equal to $(1+z)^{d_{>1}}$. This gives the recurrence relation between the cost functions:

$$C_k(z) = f_k(z) + \overline{m_1}\, (1+z)^{d - d_{>1}}\, C_{k-1}(z). \tag{5}$$

We first solve equation (5) for the standard (non-paginated) case: $c_n^k = 0$ only if $n = 0$ (empty tree); this gives:

$$f_k(z) = \sum_{n \geq 1} \binom{d}{n} z^n = (1+z)^d - 1.$$

Now, for a sequence c_k satisfying the relations

$$c_0 = 0 \text{ and, for } k \geq 1 : \quad c_k = f_k + g_k\, c_{k-1} \tag{6}$$

we have:

$$c_k = \sum_{j=0}^{k-1} f_{k-j} g_{k-j+1} \cdots g_k. \tag{7}$$

This allows us to solve the recurrence relation (5) between the $C_k(z)$, with:

$$g_{k-j} = \overline{m_{j+1}} (1+z)^{d_{>j} - d_{>j+1}}.$$

We get the generating function

$$C_k(z) = \sum_{j=0}^{k-1} \overline{m_1} \ldots \overline{m_j} \left[(1+z)^d - (1+z)^{d - d_{>j}} \right].$$

In the paginated case, $c_n^k = 0$ as soon as $n \leq b$. This gives:

$$f_k(z) = \sum_{n > b} \binom{d}{n}\, z^n = (1+z)^d - \sum_{p=0}^{b} \binom{d}{p} z^p.$$

We likewise use relations (6) and (7) to get:

$$C_k(z) = \sum_{j=0}^{k-1} \overline{m_1} \ldots \overline{m_j} \left[(1+z)^d - (1+z)^{d-d_{>j}} \sum_{p=0}^{b} \binom{d_{>j}}{p} z^p \right].$$

The extraction of $c_n^k = \frac{[z^n]C_k(z)}{\binom{d}{n}}$ yields the final result. □

5 Average cost of searching in a doubly chained tree

A straightforward implementation of multiattribute trees may be cumbersome to use, as each node on level i has up to d_{i+1} sons. *Doubly chained trees* [1][2] can be seen as an implementation of multiattribute trees by binary trees, where each node of the binary tree has for left son the first son of the corresponding node of the multiattribute tree, and for right son its first non empty brother (in the classical definition, there is also a link from each node to its father, but this is irrelevant to our analysis). The *pagination* on doubly chained trees that we shall consider is the one induced by the associated multiattribute tree, and not the one which could be defined on the binary tree. This preserves the clustering effect of pagination, which is useful in orthogonal range queries.

We shall need more knowledge about the probability law on the space of queries than in the case of multiattribute trees. Intuitively, the rank of the selected subtrees, which did not matter in a multiattribute tree (we assumed a direct access to each subtree), now appears in the cost of the query, which deals with each subtree in a specific order. As in Section 4, we assume that the intervals on the attributes A_i are selected independently by a query. We also assume that we know the probability law of the *upper* and *lower bound* of each interval, or at least the average values of these two parameters. We denote by $\overline{l_i}$ (resp. $\overline{u_i}$) the average rank of the lower bound (resp. upper bound) in D_i of the interval selected by a random query on the i-th attribute (for an integer-valued domain $D_i = [1 \ldots d_i]$, $\overline{l_i}$ and $\overline{u_i}$ are simply the average bounds of the interval selected on attribute A_i):

$$\overline{l_i} = \sum_{l=1}^{d_i} l \; p_i(l); \quad \overline{u_i} = \sum_{u=1}^{d_i} u \; p_i(u)$$

where $p_i(l)$ (resp. $p_i(u)$) is the probability that the lower bound (resp. upper bound) of the interval selected by the query on the i-th attribute is l (resp. u). We also define $m_i = u_i - l_i + 1$ and $\overline{m_i}$.

Theorem 2:
The average search cost of a random orthogonal range query on a doubly chained tree built on n *records is:*

$$c_n^k = 1 + \sum_{j=1}^{k-1} \overline{m_1}\, \overline{m_2} \ldots \overline{m_j}\, \alpha_j$$

where

- *in the non-paginated case:*

$$\alpha_j = 2\left(1 - \frac{\binom{d-d_{>j}}{n}}{\binom{d}{n}}\right) + \overline{u_{j+1}}\left(1 - \frac{\binom{d-d_{>j+1}}{n}}{\binom{d}{n}}\right);$$

- *in the paginated case:*

$$\alpha_j = 2\,(1 - \sum_{p=0}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}}{p}}{\binom{d}{n}}) + \overline{u_{j+1}}\,(1 - \frac{\binom{d-d_{>j+1}}{n}}{\binom{d}{n}})$$

Proof of Theorem 2:
The proof proceeds along the same lines as above: we derive $c^k_{u,n}(t)$, then c^k_n, and get a recurrence equation on the generating function of the c^k_n.

We decompose the query u into $u = [l_1 \dots u_1] \bullet u_{>1}$, with $[l_1 \dots u_1]$ the interval selected at the first level, and $u_{>1}$ the restriction of u to the following levels. We also write $t = (\bullet, t_1, \dots, t_{d_1})$. We first express $c^k_{u,n}(t)$as:

$$c^k_{u,n}(t) = 1 \; + c_1(t) + c_2(t)$$

where 1 comes from accessing the root of the tree, c_1 is the cost of the query at the first level, i.e. the number of sons of the root accessed, and c_2 the cost of continuing the query at the next levels, i.e. the cost of running $u_{>1}$ on the subtrees corresponding to values selected by u at the first level. Note the difference with multiattribute trees: The cost of accessing a subtree whose root is labelled by value x is equal to 0 in a multiattribute tree, and equal to the number of non-empty subtrees whose root value is less than x in a doubly chained tree. We then sum $c^k_{u,n}$ on all trees t of size n, to get:

$$c^k_{u,n} = 1 + c_1(n) + c_2(n). \tag{8}$$

$c_1(t)$ is the number of non-empty subtrees, whose root is visited by the query during the exploration of the first level. Its average value for all trees t of size n is:

$$c_1(n) = 1 + N(n, u_1)$$

where $N(n, u_1)$ is the average number of non-empty subtrees whose value on the first attribute belongs to $[1, u_1]$. We have:

$$N(n, u_1) = u_1[1 - \frac{\binom{d-d_{>1}}{n}}{\binom{d}{n}}]$$

(see [4] for the demonstration of this formula in the case where $u_1 = d_1$, which is easily extended to other values of u_1). This gives for $c_1(n)$:

$$c_1(n) = 1 + u_1 - u_1 \frac{\binom{d-d_{>1}}{n}}{\binom{d}{n}}. \tag{9}$$

$c_2(n)$ satisfies the recurrence relation of Theorem 1:

$$c_2(n) = (u_1 - l_1 + 1) \sum_{(n_i)} \frac{\binom{d_{>1}}{n_1}\dots\binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}} c_{u_{>1}, n_i}. \tag{10}$$

Substituting the values given by equations (9) and (10) for $c_1(n)$ and $c_2(n)$ in equation (8), we find the recurrence relation:

$$c^k_n \;=\; 2 + u_1 - u_1 \frac{\binom{d-d_{>1}}{n}}{\binom{d}{n}} + (u_1 - l_1 + 1) \sum_{\substack{n_1 \dots n_{d_1} \geq 0 \\ n_1 + \dots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1}\dots\binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}} c_{u_{>1}, n_1}.$$

From now on, the derivation of the generating function of the $c^k_{u,n}$ proceeds just as in the case of the multiattribute tree. Averaging over all queries, we find:

$$c^k_n = 2 + \overline{u_1} - \overline{u_1}\frac{\binom{d-d_{>1}}{n}}{\binom{d}{n}} + \overline{m_1} \sum_{\substack{n_1 \ldots n_{d_1} \geq 0 \\ n_1 + \ldots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1} \ldots \binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}} c_{n_1}.$$

This equation translates on the generating functions as:

$$C_k(z) = f_k(z) + g_k(z)\, C_{k-1}(z)$$

with

$$f_k(z) = 2\left[(1+z)^d - 1\right] + \overline{u_1}\left[(1+z)^d - (1+z)^{d-d_{>1}}\right]$$

in the standard case,

$$f_k(z) = 2\left[(1+z)^d - \sum_{p=0}^{b}\binom{d}{p}z^p\right] + \overline{u_1}\left[(1+z)^d - (1+z)^{d-d_{>1}}\right]$$

in the paginated case, and

$$g_k(z) = \overline{m_1}\,(1+z)^{d-d_{>1}}.$$

Equations (6) and (7) yield:

$$C_k(z) = \sum_{l=0}^{k-1} \overline{m_1} \ldots \overline{m_l}\left(2\left[(1+z)^d - (1+z)^{d-d_{>l}}\right] + \overline{u_{l+1}}\left[(1+z)^d - (1+z)^{d-d_{>l+1}}\right]\right)$$

in the standard case, and

$$C_k(z) = \sum_{l=0}^{k-1} \overline{m_1} \ldots \overline{m_l}\left(2\left[(1+z)^d - (1+z)^{d-d_{>l}}\sum_{p=0}^{b}\binom{d}{p}z^p\right] + \overline{u_{l+1}}\left[(1+z)^d - (1+z)^{d-d_{>l+1}}\right]\right)$$

in the paginated case, from which Theorem 2 follows. □

6 Average access cost of a random query

In this section, we give a formula for the number of leaves retrieved by a random query on a *paginated* tree. If the records are stored in secondary memory, this represents the number of accesses to this memory. The only pagination we consider on doubly chained trees being the one induced by multiattribute trees, the access cost will be the same for a multiattribute tree on n records and for its implementation as a doubly chained tree.

Theorem 3:
The average access cost of a random orthogonal range query on a paginated multiattribute or doubly chained tree built on n *records is:*

$$c^k_n = \overline{m_1}\,\overline{m_2} \ldots \overline{m_k}\frac{n}{d} + \sum_{j=1}^{k-1} \overline{m_1}\,\overline{m_2} \ldots \overline{m_j}\,(\beta_j - \beta'_j)$$

where

$$\beta_j = \overline{m_{j+1}} \sum_{p=1}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}-d_{>j+1}}{p}}{\binom{d}{n}}$$

$$\beta'_j = (\overline{m_{j+1}} - 1) \sum_{p=1}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}}{p}}{\binom{d}{n}}.$$

In the non-paginated case, we have the obvious result:

If $\overline{m_i}$ is the average size of the interval selected by the query on attribute A_i, the average access cost for a non-paginated tree is $\frac{n}{d}\prod_{i=1}^{k}\overline{m_i}$.

For a paginated tree, things are more complex.... Roughly speaking, the terms β_j correspond to the cost of retrieving unwanted records, and the terms β'_j to the gain in retrieving several records with a single page access. The access cost of a query on a pruned tree is greater than on the standard tree: we retrieve one leaf for each record which satisfies the query, and we may retrieve records which do not satisfy the query, but are in accordance with it on the first few attributes. For a page size of $b \geq 2$, the gain of retrieving up to b records satisfying the query with one access may outweight the loss of retrieving records which, when known *in extenso*, are found not to satisfy the query, and pagination may then reduce the access cost of a query.

Proof of Theorem 3:
As for Theorems 1 and 2, we establish a recurrence relation on the cost c_n^k, which we translate on the generating function $C_k(z)$. $c_0^k = 0$ and, for $1 \leq n \leq b$, $c_n^k = 1$. For $n > b$, we have:

$$c_n^k = \overline{m_1} \sum_{\substack{n_1 \ldots n_{d_1} \geq 0 \\ n_1 + \ldots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1}\cdots\binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}} c_{n_1}^{k-1}.$$

This gives:

$$C_k(z) = \sum_{p=1}^{b} \binom{d}{p} z^p + \overline{m_1} \sum_{n>b} \sum_{\substack{n_1 \ldots n_{d_1} \geq 0 \\ n_1 + \ldots + n_{d_1} = n}} \frac{\binom{d_{>1}}{n_1}\cdots\binom{d_{>1}}{n_{d_1}}}{\binom{d}{n}} c_{n_1}^{k-1} z^n$$

which is of the type of equation (5) for $f_k(z) = \sum_{p=1}^{b}[\overline{m_1}\binom{d-d_{>1}}{p} - (\overline{m_1}-1)\binom{d}{p}]z^p$. The limit case of a tree built on one attribute gives:

$$C_1(z) = f_1(z) + \overline{m_k} z (1+z)^{d_k - 1}$$

and we get:

$$C_k(z) = \overline{m_1} \ldots \overline{m_k} z (1+z)^{d-1} + \sum_{j=1}^{k-1} \overline{m_1} \ldots \overline{m_j}\, \alpha_{k-j}(z)(1+z)^{d-d_{>j}}.$$

The extraction of c_n^k completes the proof. □

7 Asymptotic formulæ

In this section, we consider the asymptotic costs obtained for a fixed underlying scheme of tree, when the sizes of the domains grow large. We shall analyse the access and search cost of a random query under the following assumptions:

The sizes d_i of the domains, the average length $\overline{m_i}$ of the intervals selected on attributes A_i, and, in the case of a doubly chained tree, the average upper bounds of these intervals remain in constant proportion. In other words, there exist sets of constants $\{\delta_i\}$, $\{\mu_i\}$ and $\{\eta_i\}$ such as, for $1 \leq i \leq k$: $d_i = \delta_i \Delta$, $\overline{m_i} = \mu_i \Delta$ and $\overline{u_i} = \eta_i \Delta$ where $\Delta \to \infty$. The number k *of attributes and the size* b *of a page are fixed. As for the size* n *of the file, we shall consider two cases:*

1. n *is fixed; it is then* $o(\Delta)$.
2. n *is of the order of* Δ*; we shall simply take* $n = \Delta$.

Theorem 4:
In case 1, where the number of records in a file remains fixed, the average search cost of a random query on a non-paginated multiattribute tree is:

$$1 + \sum_{j=1}^{k-1} \frac{\overline{m_1} \ldots \overline{m_j}}{d_1 \ldots d_j}\, n \;+\; o(1).$$

The average search cost on a non-paginated doubly chained tree is:

$$1 + \sum_{j=1}^{k-1} \frac{\overline{m_1} \ldots \overline{m_j}}{d_1 \ldots d_j} \left(2 + \frac{\overline{u_{j+1}}}{d_{j+1}}\right) n \;+\; o(1).$$

In the paginated case ($b \geq 1$), the average search cost on a multiattribute tree becomes:

$$1 + o(1)$$

and on a doubly chained tree:

$$1 + \sum_{j=1}^{k-1} \frac{\overline{m_1} \ldots \overline{m_j}}{d_1 \ldots d_j} \frac{\overline{u_{j+1}}}{d_{j+1}}\, n \;+\; o(1).$$

The average access cost on a multiattribute or doubly chained tree is:

$$\frac{\overline{m_1}}{d_1} + o(1).$$

Proof of Theorem 4:

We first approximate the coefficients $\frac{\binom{d-d_{>j}}{n}}{\binom{d}{n}}$, using Stirling's formula:

$$m! = \sqrt{2\pi m}\; m^m e^{-m} (1 + O(1/m)).$$

We get: $\frac{\binom{d-d_{>j}}{n}}{\binom{d}{n}} = 1 - n\frac{d_{>j}}{d} + O(1/\Delta^{2j})$. We then substitute this approximation in the exact formulæ of Theorems 1 , 2 and 3. □

Theorem 5:
In case 2, where the number of records in a file tends to infinity, the average search cost of a random query on a non-paginated multiattribute tree is:

$$\sum_{j=1}^{k-1} \frac{\overline{m_1} \ldots \overline{m_j}}{d_1 \ldots d_j}\, n \;+\; \overline{m_1}(1 - n/d_1 - e^{-n/d_1}) + O(1).$$

The average search cost on a non-paginated doubly chained tree is:

$$\sum_{j=1}^{k-1} \frac{\overline{m_1} \ldots \overline{m_j}}{d_1 \ldots d_j} \left(2 + \frac{\overline{u_{j+1}}}{d_{j+1}}\right) n \;+\; \overline{m_1}(1 - n/d_1 - e^{-n/d_1}) + O(1).$$

In the paginated case ($b \geq 1$), the average search cost on a multiattribute tree becomes:

$$\overline{m_1}(1 - e^{-n/d_1} e_b(n/d_1)) + O(1)$$

and on a doubly chained tree:

$$\sum_{j=1}^{k-1} \frac{\overline{m_1} \ldots \overline{m_j}}{d_1 \ldots d_j} \frac{\overline{u_{j+1}}}{d_{j+1}}\, n \;+\; \overline{m_1}(1 - e^{-n/d_1} e_b(n/d_1)) + O(1).$$

The average access cost on a multiattribute or doubly chained tree is:

$$\overline{m_1}\left(\frac{n}{d_1}\frac{m_2}{d_2}(1 - e^{-n/d_1} e_{b-1}(n/d_1)) + e^{-n/d_1}(e_b(n/d_1) - 1)\right) + O(1).$$

In these formulæ $e_b(x)$ is the truncated exponential $\sum_{p=0}^{b} x^p/p!$.

Proof of Theorem 5:

As in Theorem 4, we approximate $\frac{\binom{d-d_{>j}}{n}}{\binom{d}{n}}$ by Stirling's formula. For $j = 1$:

$$\frac{\binom{d-d_{>1}}{n}}{\binom{d}{n}} = e^{-n/d_1}(1 + O(1/\Delta))$$

and for $j \geq 2$:

$$\frac{\binom{d-d_{>j}}{n}}{\binom{d}{n}} = 1 - n\frac{d_{>j}}{d} + O(1/\Delta^{2j-2}) + O(1/\Delta^{k-1}).$$

We then substitute these asymptotic expansions in the exact formulæ .□

The asymptotic values given by Theorems 4 and 5 may be compared to the access cost on a standard tree: $\overline{m_1}\ \overline{m_2} \ldots \overline{m_k}\frac{n}{d}$. Except in the case where attribute A_i is left unspecified, we have $\frac{m_i}{d_i} < 1$, which means that the access cost of a random query on a paginated tree may well become much more important than the access cost on a non-paginated tree, as seen on the example of Section 3.

8 Applications

We give in this section consequences of Theorems 1 to 3. The search costs given here are valid for paginated and non-paginated, multiattribute and doubly chained trees, by adjusting the values of the α_j.

8.1 Single query

Corollary 1:
For a fixed query of profile $u = (m_1, \ldots, m_k)$, *and for a tree of size* n, *the average search cost is:*

$$c^k_{u,n} = 1 + \sum_{j=1}^{k-1} m_1\, m_2 \ldots m_j\, \alpha_j.$$

The average access cost on a paginated tree is:

$$c^k_n = m_1 m_2 \ldots m_k \frac{n}{d} + \sum_{j=1}^{k-1} m_1 m_2 \ldots m_j\, \gamma_j$$

with

$$\gamma_j = m_{j+1} \sum_{p=1}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}-d_{>j+1}}{p}}{\binom{d}{n}} - (m_{j+1}-1) \sum_{p=1}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}}{p}}{\binom{d}{n}}.$$

8.2 Partially specified search with probabilities

We assume here that, for $1 \le i \le k$, attribute A_i is either uniquely determined, or left fully unspecified. It has a probability p_i of being specified (i.e. taking any fixed value of the domain D_i), and $1 - p_i$ of not being specified (i.e. any value of D_i is admissible).

Corollary 2:
For a random partial match query on a tree of size n, *the average search cost is:*

$$c^k_n = 1 + \sum_{j=1}^{k-1} (p_1 + (1-p_1)d_1) \ldots (p_j + (1-p_j)d_j)\, \alpha_j.$$

The average access cost on a paginated tree is:

$$c^k_n = (p_1 + (1-p_1)d_1) \ldots (p_k + (1-p_k)d_k)\, \frac{n}{d} + \sum_{j=1}^{k-1} (p_1 + (1-p_1)d_1) \ldots (p_j + (1-p_j)d_j)\, \gamma_j$$

with

$$\gamma_j = (p_{j+1} + (1-p_{j+1})d_{j+1}) \sum_{p=1}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}-d_{>j+1}}{p}}{\binom{d}{n}} - (1-p_{j+1})(d_{j+1}-1) \sum_{p=1}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}}{p}}{\binom{d}{n}}.$$

8.3 Memory requirements

The average memory needed to store a tree (which can be seen as a directory giving access to the leaves) is easily derived from the general formulæ. The storage size has two components: size of the directory, which is proportional to the number of internal nodes of the tree, and size of the leaves, where the records are stored. We can express both as the search and access cost of a query where *all* attributes are left unspecified.

Corollary 3:
The average number of internal nodes of a (standard or paginated) tree on n *records is:*

$$c_n^k = 1 + \sum_{j=1}^{k-1} d_1 \dots d_j \, \alpha_j .$$

The average number of leaves of a paginated tree on n *records is:*

$$n - \sum_{j=0}^{k-1} d_1 \dots d_j \gamma_j$$

with

$$\gamma_j = (d_{j+1} - 1) \sum_{p=1}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}}{p}}{\binom{d}{n}} - d_{j+1} \sum_{p=1}^{b} \frac{\binom{d-d_{>j}}{n-p}\binom{d_{>j}-d_{>j+1}}{p}}{\binom{d}{n}}.$$

References

[1] A.F. Cardenas, J.P. Sagamang: Modeling and analysis of data base organization: the doubly chained tree structure. *Information Systems,* Vol. 1, 57-67 (1975).

[2] A.F. Cardenas, J.P. Sagamang: Doubly chained tree data base organization-analysis and design strategies. *The Computer Journal,* Vol. 20, No 1, 15-26 (1977).

[3] P. Flajolet, C. Puech: Partial match retrieval of multidimensional data. *Journal of the A.C.M.,* Vol. 33, No 2, 371-407 (April 1986).

[4] D. Gardy, C. Puech: On the size of projections: a generating function approach. *Information Systems,* Vol. 9, No 3/4, 231-235 (Oct. 1984).

[5] V. Gopalakrishna, C.E. Veni Madhavan: Performance evaluation of attribute-based tree organization. *A.C.M. Transactions On Database Systems,* Vol. 6, No 1, 69-87 (March 1980).

[6] R.L. Kashyap, S.K.C. Subas, S.B. Yao: Analysis of the multiple-attribute-tree database organisation. *I.E.E.E. Transactions on Software Engeenering,* Vol. SE-3, No 6, 451-467 (Nov. 1977).

[7] C. Puech: *Méthodes d'analyse de structures de données dynamiques.* Thèse d'Etat, Orsay (France), Avril 1984.

[8] P. Svensson: On search performance for conjunctive queries in compressed, fully transposed ordered files. *Proc. Fifth International Conference on Very Large Data Bases,* Rio de Janeiro, Brazil, 155-163 (3-5 Oct. 1979).

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique